Research online Publishing www.ronpub.com

Open Access

Open Journal of Databases (OJDB)
Volume 5, Issue 1, 2018

www.ronpub.com/ojdb
ISSN 2199-3459

Effectiveness of NoSQL and NewSQL Databases in Mobile Network Event Data: Cassandra and ParStream/Kinetic

Petri Kotiranta, Marko Junkkari

Faculty of Natural Sciences, Computer Sciences, University of Tampere, FI-33014, Finland,
kotiranta.petri.S@student.uta.fi, marko.junkkari@uta.fi

ABSTRACT

Continuously growing amount of data has inspired seeking more and more efficient database solutions for storing and manipulating data. In big data sets, NoSQL databases have been established as alternatives for traditional SQL databases. The effectiveness of these databases has been widely tested, but the tests focused only on key-value data that is structurally very simple. Many application domains, such as telecommunication, involve more complex data structures. Huge amount of Mobile Network Event (MNE) data is produced by an increasing number of mobile and ubiquitous applications. MNE data is structurally predetermined and typically contains a large number of columns. Applications that handle MNE data are usually insert intensive, as a huge amount of data are generated during rush hours. NoSQL provides high scalability and its column family stores suits MNE data well, but NoSQL does not support ACID features of the traditional relational databases. NewSQL is a new kind of databases, which provide the high scalability of NoSQL while still maintaining ACID guarantees of the traditional DBMS. In the paper, we evaluation NEM data storing and aggregating efficiency of Cassandra and ParStream/Kinetic databases and aim to find out whether the new kind of database technology can clearly bring performance advantages over legacy database technology and offers an alternative to existing solutions. Among the column family stores of NoSQL, Cassandra is especially a good choice for insert intensive applications due to its way to handle data insertions. ParStream is a novel and advanced NewSQL like database and is recently integrated into Cisco Kinetic. The results of the evaluation show that ParStream is much faster than Cassandra when storing and aggregating MNE data and the NewSQL is a very strong alternative to existing database solutions for insert intensive applications.

TYPE OF PAPER AND KEYWORDS

Short Communication: *performance evaluation, data storage, data aggregation, insertion, telecommunication, NoSQL, NewSQL, Cassandra, ParStream, Kinetic, Mobile Network Event, MNE data*

1 INTRODUCTION

In telecommunication, ever bigger data sets must be manipulated because the number of transactions and the amount of data associated with the transactions increase constantly. International and national laws and standards determine what kind of data must be stored about a single transaction, and thus they determine the structure of data. Due to the increasing amount of Mobile Network Event (MNE) data it is essential to investigate possible solutions to manipulate MNE data. NoSQL [13][14] and NewSQL [14][26] databases are modern

solutions to manipulate big data sets. In this present study, we evaluate and compare the efficiency of NoSQL databases and NewSQL databases in storing and querying MNE data. Applications of MNE data are insert intensive, and thus the main focus of the evaluation is to compare the storing speed of databases. More generally, our research question is whether new database solutions bring additional value compared to existing legacy SQL based solutions.

The traditional SQL databases are usually designed to be operated on one server node. This is the way they can offer ACID (Atomicity, Consistency, Isolation, and Durability) properties. However, the drawback of this feature is the lack of horizontal scalability. Depending on the implementation, clustering is possible. For example, in Oracle it is possible to divide the database tables into different server nodes. However, NoSQL databases enable to spread the data around a cluster per data row based on the primary key. This is why there has been the demand for NoSQL databases as they offer more horizontal scalability. They also offer simpler data models that may be more efficient than SQL in certain use cases. NewSQL is a class of SQL database systems, which seek to achieve high performance and scalability of NoSQL while still guaranteeing the ACID properties of traditional DBMS. By comparing between NoSQL databases and NewSQL databases, we aim to find out whether it is possible to combine the best sides of the SQL and NoSQL databases and still perform well in our use case.

Among NoSQL databases, column family stores and document stores are structurally suitable for manipulating NME data. We selected column family store Cassandra because its insertion speed is efficient and its data files take less storage space than JSON based document store implementations. Among NewSQL databases we selected ParStream [7] because it is suitable for managing MNE data. For example, it supports the geo-distributed database solution that is essential for MNE data. As of the summer 2018, after ParStream was acquired by Cisco (www.cisco.com) that is a market leader in the areas of IT and network. Cisco integrates it into Cisco Kinetic system and does not offer ParStream as a stand-alone product any more [10], but Cisco still provides all documents on ParStream online [8]. The functionality of ParStream is now a part of Cisco Kinetic, and hence we also refer to this database by ParStream/Kinetic. In the test setting we use original databases because during our test periods for MNE data we had only the license of ParStream but not Kinetic.

MNE data consists of different reports that are generated by telecommunications traffic. The most common report is the RAB (Radio Access Bearer) report that is sent when a radio access bearer is created. A radio access bearer provides a connection between a user equipment and a network service. It is created practically every time when a user equipment, for example a mobile phone, tries to connect to a mobile phone network through a base station. These reports consist of structured data that have values representing several metrics from the base station and information about the user equipment such as IMEI (International Mobile Equipment Identity) and IMSI (International Mobile Subscriber Identity). These, in turn, contain coded information on the country, networks and the route through which a mobile phone plan has been connected. For example, IMSI is a 64-bit field typically represented as a 15 digit number where first three digits determine a country, and a mobile network code and a mobile subscription identification number follow.

For a mobile phone plan, there is information on the country, networks and the route through which a mobile phone plan has been connected. A connection involves information on different kinds of area codes and the route through which the connection is formed. A network station carries its own information. All this information is collected into a RAB report that is stored in the context of a MNE event. Depending on a version, a RAB report contains about 100 data entries, but one entry may contain a value having different kind of coded data.

In a typical scenario, a huge amount of RAB reports must be stored during a short period of time. This happens especially during rush hours when many user equipment requests RAB. Therefore, the storing speed plays the most essential role in manipulating RAB reports. Most of the data will not be utilized, but the storing is necessary for tracing possible problems or tracking calls in serious criminal cases. The RAB reports can also be used analyzing the load of a network in a specific area. Therefore, aggregation queries are essential when analyzing the reports. Furthermore, pattern matching queries such as ‘like’ are needed to isolate parts of the codes (e.g. the country code of IMSI).

In the evaluation, we simulate real world multi-columned data from the area of telecommunication, the data storing and aggregation performance of the NewSQL-like ParStream database and the NoSQL Cassandra database are evaluated over different amount of MNE data using different number of threads. The results of the evaluation show that ParStream is dramatically faster than Cassandra in storing data and it benefits from increasing the number of threads. ParStream also outperforms the traditional SQL solution in the insertion speed of data. The efficiency of aggregation queries depends on the column on which the query is focused. If aggregation queries do not focus on any specific columns, ParStream is notable faster than Cassandra. We also conclude that Cassandra does not support pattern matching queries that are essential for manipulating NME data.

In this study work, we compare NoSQL and NewSQL-like databases whereas existing studies focus mainly on NoSQL databases. To our best knowledge, no

research results have been published on efficiency evaluation of either ParStream or Kinetic and on the performance comparison between NoSQL databases and NewSQL databases.

The rest of the paper is organized as follows. In Section 2, we perform a literature review on efficiency studies of NoSQL databases. Section 3 investigates different database models in order to find suitable databases, which will be used in this study to evaluate the performance of databases for MNE applications. In Section 4, we introduce the content of the Radio Access Bearer (RAB) reports. RAB reports are the Mobile Network Event (MNE) data and are be used in the evaluation. Section 5 describes the generation of RAB data and the evaluation setting. The results of evaluation are presented in Section 6. Section 7 discusses the evaluation results and investigates further research questions. Finally, conclusions are given in Section 8.

2 RELATED WORK

There has been a large amount of research on the performance of the NoSQL databases [1][2][4][9][11] [17][18][19][20][25]. Yahoo! Cloud Serving Benchmark [9][27] is the most popular testing environment for key-value data. More complex data are used in benchmarking document stores with SQL databases in [23] and 0. Oliveira and Bernardino [21] have compared NewSQL databases MemSQL and VoltDB using the TPC-H test set that is also a more complex data set containing several tables and their mutual relationships. We focus on the studies where column family stores are compared with other databases. In the following, we present the latest performance tests where Cassandra is compared with column family stores HBase, Hypertable, document stores MongoDB, Couchbase, RavenDB, CouchDB, key-value databases Aerospike, Redis, multimodel database OrientDB and relational database MS SQL Express.

In the Datastax study [11], Cassandra version 1.1.6, HBase version 1.1.1 and MongoDB version 2.2.2 have been compared. Yahoo! Cloud Serving Benchmark was used as a test tool. Load, read, write and scan tests were made with different stress levels and different amounts of cluster nodes. Read, insert, update and scan latency were also tested. Cassandra had clearly the best performance among the databases. Especially, when the amount of cluster nodes was increased Cassandra was much ahead leaving HBase second and MongoDB third.

Nelubin and Engber [20] compared Cassandra, MongoDB, Couchbase and Aerospike. In their study, the performance of the databases was compared using Yahoo! Cloud Service Benchmarking Tool. Databases were compared for insertion throughput, maximum throughput and latencies in balanced workload (50% write and 50% read) and read heavy workload (95% read and 5% update) in SSD (Solid State Drive)-backed and in-memory datasets. The tests measured raw key-value performance of the databases. In these tests, Aerospike and Couchbase had clearly better performance compared with Cassandra and MongoDB. Aerospike outperformed Couchbase in read-heavy workloads and Couchbase outperformed Aerospike in balanced read-write workloads. One of the reasons for the good performance of Aerospike was that it had been well optimized for SSD disks that were used in this test. Both Aerospike and Couchbase are designed for key-value based queries and these databases were expected to perform better than more complex Cassandra and MongoDB. However, pure key-value performance is not what we are looking for as MNE applications usually require more complex queries.

Li and Manoharan [19] compared MongoDB version 1.8.5, RavenDB version 960, CouchDB version 1.2.0, Cassandra version 1.1.2, Hypertable version 0.9.6, Couchbase version 1.8.0 and MS SQL Express version 10.50.1600.1. The study tested instantiating a bucket of key-value pairs, reading values behind keys, creating and updating key-value pairs, deleting key-value pairs and fetching all the keys. RavenDB, Hypertable and MongoDB were the fastest whereas CouchDB, Couchbase and SQL Express were the slowest in creating the bucket. The read performance list of databases from the fastest to the slowest was as follows: Couchbase, MongoDB, SQL Express, Hypertable, CouchDB, Cassandra and RavenDB. With write performance, the corresponding list was Couchbase, MongoDB, Cassandra, Hypertable, SQL Express, RavenDB and CouchDB, and with delete performance Couchbase, MongoDB, SQL Express, Cassandra, Hypertable, CouchDB and RavenDB. In fetching all the keys, the test observation was that all the databases fetch keys quickly except CouchDB. SQL Express was the fastest for doing this operation.

One of the interesting findings of this study was that traditional database MS SQL Express performed better than some of the NoSQL databases. Thus, although NoSQL databases should perform better in key-value based queries compared with traditional databases, they do not always perform better than traditional SQL databases. There was only a small correlation between performance and data models. RavenDB and CouchDB were not good in read, write and delete operations. Couchbase and MongoDB were overall the fastest for read, write and delete operations. Cassandra was slow in read operations but good in write and delete operations. Anyway, Cassandra had the best performance among column family stores.

Abramova and others [2] compared Cassandra version 1.2.1, HBase version 0.94.10, MongoDB version 2.4.6, OrientDB version 1.5 and Redis version 2.6.14. Among these databases OrientDB can be used as

Table 1: Top four ranking of NoSQL database performance tests

Test	1.	2.	3.	4.
Klein et al. [17]	Cassandra	Riak	MongoDB	-
Datastax [11]	Cassandra	HBase	MongoDB	-
Nelubin & Engber [20]	Couchbase	Aerospike	MongoDB	Cassandra
Li & Manoharan [19]	Couchbase	MongoDB	Cassandra	Hypertable
Abramova et al. [2]	Redis	Cassandra	HBase	MongoDB

a document store and a graph database. Databases were tested with Yahoo! Cloud Serving Benchmark program. Read and write operations were tested with 600,000 records. Tests focused on comparing the execution speed of get and put operations with different workloads of read and update operations. Redis was clearly the fastest of the tested databases, Cassandra the second fastest, HBase third, MongoDB fourth. The slowest was OrientDB when comparing the overall execution time of workloads. One of the reasons for the poor performance of the OrientDB was that it keeps records in the disk rather than loading them into memory. Other reason mentioned was that OrientDB took more resources than what was available in the test environment. Abramova and others [2] divide NoSQL databases into two categories: those that are good in read operations and those that are good in update operations. MongoDB, Redis, and OrientDB belong to the first category, whereas Cassandra and HBase belong to second category. Cassandra again possessed the best performance among column family stores and it performed well especially in write operations. Therefore, Cassandra is a strong alternative for insert intensive applications.

Klein and others [17] have compared MongoDB version 2.2, Cassandra version 2.0 and Riak version 1.4. A modified version of the Yahoo! Cloud Serving Benchmark framework was used for testing. Tests measured the throughput of read-only, write-only and read/write workloads, and read and write latencies. Each test was run three times with different number of threads. The performance of Cassandra was clearly best in the read and write tests when the number of threads was increased. Riak had the second best performance and third was MongoDB. On the one hand, Cassandra had the biggest delay in read and write operations whereas Riak was 5 times faster and MongoDB was 4 times faster. The reason for the better performance of Cassandra was that its hash based sharding was much more efficient than the sharding of MongoDB. On the other hand, the indexing features of Cassandra enabled fast queries. Furthermore, the peer to peer based architecture facilitated efficient coordination of read and write operations between different nodes. From the perspective of the present study, the results are interesting because we also run tests using the different number of threads.

All the mentioned tests, where Cassandra participated, were key-value oriented. Yahoo! Cloud Service Benchmarking tool was used in many of the tests and this tool measures get and put performance with different loads. As key-value stores are well optimized for these kinds of queries, they had the best performance. However, among column family stores Cassandra performed best and therefore we selected Cassandra for testing multicolumn MNE data. Table 1 summarizes the results of the performance tests. Our test setting differs from above-mentioned evaluation in two ways. First, we use real words multi-columned data and second, we evaluate not only NoSQL databases but also compare NoSQL with NewSQL databases.

3 REVIEWING DATABASES FOR TESTING

In many of the current MNE applications, the solutions are based on traditional SQL databases. As MNE data has a heavy demand for insertion performance, we are interested in NoSQL and NewSQL solutions for MNE data. In this part, we reviewed different NoSQL data models and aimed to find suitable databases in order to investigate if traditional SQL database could be replaced with the NoSQL and NewSQL solutions for the needs of MNE data applications.

NoSQL databases are intended for big data sets and their organization is not based on the relational model. The query capabilities of NoSQL solutions are different in comparison with traditional SQL solutions, and this must be taken into account when selecting a suitable NoSQL database solution for MNE data that is multi-column, structurally predetermined and contains a low number of relationships. The graph-based, column family stores, document stores and key-value stores are different types of NoSQL databases [12].

Key-value store model is very simple and does not support as diverse queries as SQL does. In multicolumn data, SQL based solutions are difficult to replace with simple key-value store solutions because the keys to query must be known beforehand. Graph databases are neither a suitable choice for MNE data because they are designed for data that has a great number of relationships. Instead, document stores and column family stores are suitable for MNE data because both of

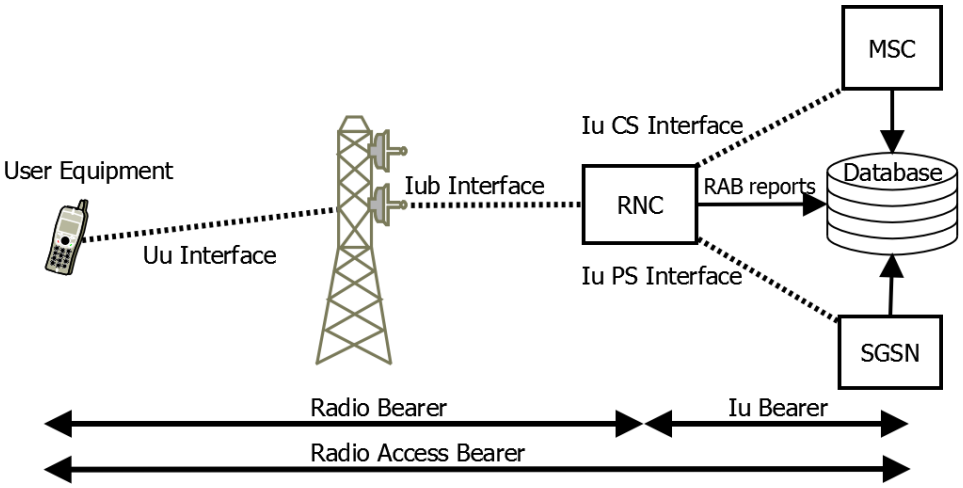

Figure 1: Radio Access Bearer in UMTS system

them contain structured data under key and support queries to different attributes of data. Document stores are based on JSON format. However, one of the drawbacks of JSON is that it consumes space because of the structure definition of the JSON standard. Column family stores, like Cassandra and Hypertable, support SQL-like query language CQL (Cassandra Query Language) [6] and HQL (Hibernate Query Language) [16]. However, the expression power of these languages is limited in comparison with the standard SQL. For example, join operations are not supported.

In column families, data also take less space compared with document stores. This is due the fact that column definitions take less space on disc than JSON structure definitions. Thus, among NoSQL databases, the column family store model suits best for the needs of MNE data. We tested space consumption of 1,000,000 RAB reports in one of the most popular document stores, MongoDB and one of the most popular column family stores, Cassandra. The size of RAB reports as a MongoDB collection was 4.53 GiB and as a Cassandra column family was 1.18 GiB. As the system might have to store billons of these reports, we considered Cassandra more space efficient.

Among column family stores, Cassandra has clearly the highest ranking in DB-Engines [12]. In Cassandra previously mentioned amount of reports consume around as much space as in a SQL database. As reviewed in Section 2, Cassandra has the best performance among the column family stores. The way Cassandra stores the data should suit insertions well. When inserting data, Cassandra just appends the data into commitlog and memtable. The operation is simple and thus insertion operations should be efficient. Therefore, we select Cassandra as a representative of column family stores in our testing.

NewSQL solutions are a new group of databases that aim to provide the best sides of the two kinds of databases: the high scalability of NoSQL and the ACID features of traditional relational databases. There are three main categories of NewSQL databases: 1. New database solutions that have been written from scratch; 2. MySQL based storage engines; 3. Pluggable solutions for existing databases that aim to provide more scalability [26]. We chose ParStream to represent a NewSQL database with properties from both SQL and NoSQL databases. Apart from supporting traditional SQL queries, ParStream also provides horizontal scalability that is not offered by traditional SQL solutions. The ParStream database handles data in partitions, i.e. a table can be partitioned based on chosen partitioning columns. This way ParStream enables very fast querying as it can exclude irrelevant partitions by using bitmap indexing [24].

Furthermore, ParStream is one of the newest NewSQL like database with possibility to install a geo-distributed analytics server. This is an essential feature for geo-distributed telecommunication architecture. Although we do not investigate geo-distributed analytics in the present study, this was still one reason for selecting ParStream out of other NewSQL solutions. To our best knowledge, no previous research on ParStream exists, so we aim to show in the present research how this kind of database performs against Cassandra.

4 TEST DATA – RAB REPORTS

In a performance evaluation, it is essential that the test data corresponds to real data [3]. Our test data structurally and in content corresponds to Radio Access Bearer (RAB) reports. Radio Access Bearers are used when a user equipment, for example a mobile telephone, connects to a mobile network. RAB guarantees bandwidth for different kinds of communication that a

mobile equipment does in the network. Different sorts of RABs are used for different types of communication. For example, conversational speech RABs are used for normal telephone calls. These RABs guarantee 12.2 kbps bandwidth for speech. Web browsing and email sending activities use interactive packet switched RABs that guarantee 384 kbps downlink and 64 kbps uplink. Many other kinds of RABs also exist for different kinds of connection.

Figure 1 illustrates the components of RAB in UMTS system. As can be seen from the picture, Radio Access Bearer consists of Radio Bearer and Iu Bearer. Radio Bearer is created between a user equipment and Radio Network Controller (RNC). Radio Network Controller is an element that is responsible for managing resources between a radio network and a core network. Iu Bearer is created between RNC and Mobile Switching Centre (MSC) in a circuit switched core network and Serving GPRS Support Node (SGSN) in a packet switched core network. MSC routes voice calls and SMS messages to the circuit switched network. SGSN works similarly for packet switched data.

Network operators are interested in monitoring the activities that occur in the network. This is why a network element, such as RNC, sends reports when RAB is created. The RAB report is a part of commonly accepted 3GPP (3rd Generation Partnership Project) specifications [28]. Network event monitoring systems are used to analyze the reports. All the created RAB reports are usually collected into a system database. In Figure 1, data are collected from the RNC, MSC and SGSN elements into a database. During rush hours, when a lot of RABs are established, a huge number of reports might be sent, so it is very important for the database to perform fast enough to handle all these reports.

The content of the RAB reports varies in some extent and depends on the network element that sends them. A RAB report typically contains information on the user equipment that requests RAB and technical information related to a base station and connection. In our case, a RAB report contains information about user equipment such as International Mobile Subscriber Identity (IMSI), International Mobile Station Identity (IMEI), Mobile Station International Subscriber Directory Number (MSISDN) and an IP address. There is also much information related to base stations and connections. This information includes start and stop base stations and their Cell ID (CID), UTRAN Cell ID (LCID), Mobile Country Code (MCC), Mobile Network Code (MNC), Special Area Code (SAC) and Location Area Code (LAC). Further, RAB reports contain information about many other kinds of connection and possible failure. The RAB reports that we used in our test contain 96 columns.

5 EVALUATION SETUP

A Java program was implemented to generate the test data that structurally correspond to real RAB reports used in the UTRAN network elements. More concretely, the program creates an array object with 96 columns for a RAB report, which contains 88 columns of 32-bit integer types, one timestamp and seven string types. The integers are generated using the random class of Java with the ranges of real values what are used in the actual reports. Some string types, such as IMSI (International Mobile Subscriber Identity) and URL (Uniform Recourse Locator), are selected from the real data and other data (e.g. IPv4 -Internet Protocol version 4) are generated at random. The timestamp is the time when the array is inserted into the database. All data are stored into a table called *networkdata*. In Appendix A, the code for generating the test data is given. Three columns were indexed that correspond to real indexing needs for typical use cases of MNE data.

The computers used in the tests were HP ProLiant DL380 Gen9 Server. The used operating system is Red Hat Enterprise Linux version 6.5. Datastax Cassandra version 2.2 and ParStream version 3.3.4 were installed. Cassandra driver version 2.1.5 was used to insert data into Cassandra and Java Streaming Import API version 3.3.4 was used to insert data into ParStream. We used only one node installation of the both databases. The HP ProLiant DL380 Gen9 Server had following hardware setting:

- 2 x Intel Xeon E5-2667 v3 CPU @ 3.20GHz
- 8 cores, 16 threads
- 64-bit memory technology
- L1 cache 512 KB
- L2 cache 2048 KB
- L3 cache 20480 KB
- 32 GB memory for each processor @ 2133 MHz.

In the test setting, the servers contain only necessary programs and no unnecessary external load existed. For inserting data into databases, the Java program utilizes a for-loop that iterates through a list of array objects. The values of objects are randomly generated following the database structure presented in Appendix A. In our tests, the initial size of data was 10,000 rows. The for loops either keep looping trough the list until certain amount of time has passed or certain amount of rows are inserted. So as the same pre-generated list of rows is looped trough many times, some of the values can be duplicates. In our tests, this is not important as we only are interested in the insertion and query performance. The Java program can also insert values in multiple threads and the amount of threads can be selected.

The tests were run in a single node for achieving comparability with the existing architecture that is designed for a single node database.

Table 2: Average insertion rate in rows per second for twelve five-minute runs

Number of Threads	ParStream	Cassandra
1	30459	1132
2	56393	5454
3	79114	4918
4	101108	4491
5	117278	4032
6	130448	3619
7	151606	3161
8	167868	2885
9	197233	2492
10	220393	2187
11	242622	2063
12	241366	2029

6 EVALUATION

In this study, we evaluate the performance of a NoSQL database (Cassandra) and a NewSQL database (ParStream), and aim at answering the question whether new database solutions bring additional value compared to existing legacy SQL based solutions. Since applications of MNE data are insert intensive, this evaluation focuses on the efficiency of storing and aggregation query processing.

6.1 Storing Speed

The efficiency of storing data is essential in telecommunication because during a short time period a large amount of data may be inserted into a database. The number of threads is a typical way to increase storing speed.

We first compare ParStream and Cassandra using a single thread. The average storing speed in Cassandra was about 12,500 rows per second and in ParStream about 40,000 rows per second. In other words, the storing speed was over three times faster in ParStream than in Cassandra. During the two-hour period of testing the difference stayed linearly the same.

In order to test the impact of the number of threads in storing data, we made twelve five-minute runs using different number of threads with both databases. The results are given in Table 2 and illustrated in Figure 2. In ParStream, increasing the number of threads also increased the storing speed. Cassandra, instead, did not benefit notably from increasing the number of threads. The difference of the maximum storing speeds was 44 times bigger in ParStream than in Cassandra. Based on the performance of relational database solutions, we choose the insertion speed of 210,000 rows per second as the performance baseline. Cassandra did not exceed the performance baseline even with the maximal number of threads, whereas ParStream exceeded the performance baseline when using ten or more threads.

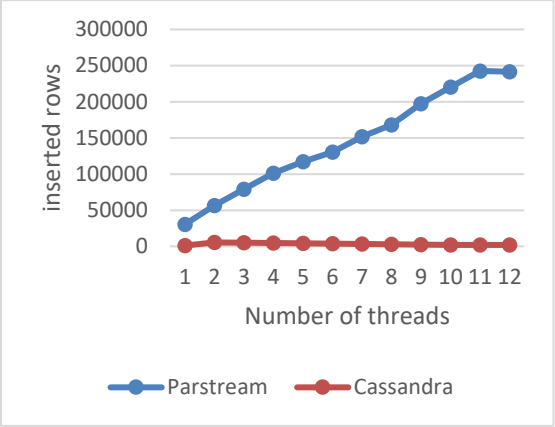

Figure 2. Average insertion rate in rows per second with different number of threads

6.2 Querying Speed

In MNE applications, it is often needed to find the amount of certain reports. Thus, we chose a count query for testing the aggregation efficiency of databases and the count query is currently used by existing SQL based applications. The count function is heavy for databases to process. It is supported by both ParStream and Cassandra and we were thus able to compare them and see the differences in performance. In Cassandra, the use of a column counter is recommended to keep track of the amount of records. Counter columns with multiple nodes may involve consistency issues.

The first query is a basic aggregation query where the number of all the rows is calculated.

Query 1: SELECT COUNT(∗) FROM *networkdata*

We tested the speed of the query over different amount of data. The tested data sets consist of half million, one million, ten million and hundred million rows. The results are given in Table 3. In all the cases, ParStream was dramatically faster than Cassandra. For example, in case of the half million rows, Cassandra performed the query in 36 seconds whereas ParStream used only four milliseconds. In the case of hundred million rows, Cassandra could not finish the execution of the query during 30 minutes whereas ParStream used only 16 seconds.

It is also worth noting that running the same query twice improved the execution time in ParStream. In the

Table 3: Execution time of Query 1

Number of Rows	Cassandra	ParStream
500,000	36 seconds	0.004 seconds (1st run) 0.002 seconds (2nd run)
1,000,000	1 min 24 seconds	0.007 seconds (1st run) 0.0035 seconds (2nd run)
10,000,000	11 min 12 seconds	0.067 seconds (1st run) 0.031 seconds (2nd run)
100,000,000	Timeout (30 minutes)	16 seconds (1st run) 0.35 seconds (2nd run)

cases of half, one and 10 million rows, the time halved. In the case of hundred million rows, the processing time decreased from 16 seconds to less than half seconds. This is due to the fact that in the first querying the bitmap is loaded into the central memory, and thus it is immediate in use in the second query processing. Cassandra does not benefit from repetitively querying.

The above results do not mean that Cassandra is inefficient in general. If a query is focused on the column that is part of the primary key, Cassandra is efficient. Query 2 represents the query type where an attribute is exactly valuated. We consider the querying efficiency related to the role of a valuated attribute. The test data contain a hundred million rows.

Query 2: SELECT COUNT(∗) FROM $networkdata$ WHERE $attribute = 44755149$

When the attribute belongs to a column that is a part of the primary key in Cassandra, and the attribute is partitioned and bitmap indexed in ParStream, there is no significant difference between the query performances of the two databases. Execution times of two databases were less than 0.01 seconds. If the attribute is indexed but not partitioned in ParSrteam, then the query was performed in 18 seconds. If the attribute is neither indexed nor partitioned in ParSrteam, the query required more than 40 seconds. We cannot run the query with these settings in Cassandra, because non-indexed attributes cannot be queried by Cassandra.

The last query is string matching, which is supported by ParStream but not by Cassandra. In Query 3, '515' is an area code that is the initial code of IMSI (the International Mobile Subscriber Identity).

Query 3: SELECT COUNT(∗) FROM $networkdata$ WHERE imsi LIKE $'515\%'$

In testing Query 3, the database contained ten million rows. ParStream performed the string matching query in 10 seconds when the bitmap was not used, whereas when using the index, the query required 28 seconds. This means that bitmap indexing does not increase performance for every type of queries.

7 DISCUSSION

We evaluated the insertion performance and the aggregation capability of ParStream and Cassandra databases over MNE data in the previous section. The results of the evaluation show that ParStream was much more efficient in insertion speed compared with Cassandra.

In order to find out why Cassandra had an inferior insertion speed, we performed more investigations on its performance by changing the test setting. We changed durable writes feature to off-state that bypasses commitlog of Cassandra. Cassandra appends the data first into commitlog-file and then takes it into in-memory memtable. If memtable is full, data are stored into data files. However, setting durable writes to off-state did not have any impact on insertion speed. We also tested the effect of different sizes of the memtable, but this did not improve insertion performance either.

ParStream seems to be better optimized for parallel insertions than Cassandra. There are differences of how the data is stored and the databases are implemented. Cassandra is programmed in Java and cannot be optimized as efficiently as C/C++ program. Cassandra stores data as key-value pairs. All the values are stored with their corresponding key. When more columns were indexed, insertion speed seemed decreasing. Thus, writing data as key-value pairs seems to be one of the factors that decrease performance.

In the aggregation test, Cassandra was clearly inferior to ParStream as well. When executing the count function of CQL (Cassandra Query Language) [6], Cassandra reads through all the rows in the database and the operation is very slow. If an aggregation query is modified such that a condition for an indexed key is inserted, both of Cassandra and ParStream show equally good performance when the queried value belongs to an indexed column. This was to be expected. Cassandra fetches the columns quickly with a right row key and ParStream takes advantage of partitioning and bitmap indexing. If the column is not partitioned, ParStream cannot exclude irrelevant partitions from the query. This

is why the query takes a longer time to finish. When using bitmaps the querying time will be halved.

Query 3 was executed just for ParStream as Cassandra does not support bitmap indexing and CQL does not support LIKE-operation. The query demonstrates that bitmap indexing does not always provide better performance. This query was faster when bitmap indexing was not used. Seeking the values that match a like pattern seems to be the weakness of the bitmap indexing in ParStream. Query 3 expresses also a general problem in comparing different databases by complex queries. Namely, if a database or a query language does not support a query type, the comparison cannot be executed. This is one reason for developing SQL++ [22] that gives a similar interface to relational databases and NoSQL databases. On the other hand, the development of the query languages of NoSQL databases is in progress and, thus, the SQL++ interfaces will be developed.

Although ParStream seems to be overwhelming in data insertion and aggregation over NME data, an open question is the efficiency of ParStream in general. For that, ParStream should be tested in different data sets and compared with other NoSQL and NewSQL databases. Yahoo! Cloud Service Benchmarking tool would get results that will be in line with the results released in NoSQL databases. TCP-H and TCP-BB [5] are public data environments with structurally more complex data. The ParStream has been acquired by Cisco and integrated into Cisco Kinetic distribute system in the summer 2018, and thus further tests should be focused on Kinetic. During our test periods we had only the license of ParStream but not Kinetic. In general, a similar test setting can be repeated with Kinetic, and similar results could be expected with Kinetic.

8 CONCLUSIONS

In this work, we investigate the applicability of NoSQL and NewSQL databases for storing and querying Mobile Network Event (MNE) data. Structurally, column family stores and document stores are suitable for MNE data, but the storage format used by document stores consumes space. Therefore, among NoSQL databases we selected the column family store databases for performance evaluation. We tested the performance of two databases over MNE data: Cassandra and ParStream. Cassandra is a column family store database and it is known as a very efficient solution for big data sets. ParStream is a NewSQL like database for which no test results have so far been published.

MNE applications are insert intensive and therefore the efficiency of storing data is essential in MNE applications. In terms of a single thread, ParStream was three times faster than Cassandra. When increasing the number of threads, both databases enhanced their storing performance, but ParStream increased obviously more. The difference of the maximum storing speed was 44 times bigger in ParStream than in Cassandra. In the test setting tree columns were indexed. This seems to be the essential reason for the huge difference in the performance. In an additional test, we found that if only one column is indexed, ParStream was six times faster than Cassandra.

In aggregation querying, ParStream was dramatically faster than Cassandra. When a query is focused on a key attribute, no difference between the databases was found. So far Cassandra is known as one of the column family stores with best performance especially for write operations. However, with insert-intensive MNE applications, ParStream is very efficient compared to Cassandra. The functionality of ParStream is now a part of Cisco Kinetic and similar results could be expected with Cisco Kinetic

The study presented in this paper clearly indicates that the new kinds of database technology can clearly bring performance advantages over legacy database technology and offer a very strong alternative to existing solutions.

REFERENCES

[1] Y. Abubakar, T.S. Adeyi and I.G. Auta, “Performance evaluation of NoSQL systems using YCSB in a resource austere environment,” *Int. J. of Appl. Inf. Systems*, vol 7, pp. 23-27, 2014.

[2] V. Abramova, J. Bernardino, and P. Furtado, “Which NoSQL database? A performance overview,” *Open J. of Databases (OJDB)*, vol. 1, no. 2, pp. 17-24, 2014. [Online]: http://nbn-resolving.de/urn:nbn:de:101:1-201705194607

[3] A. Alexandrov, C. Brücke and V. Markl, “Issues in big data testing and benchmarking,” in *Proceedings of Sixth International Workshop on Testing Database Systems*, 2013.

[4] S. Bushik, “A vendor-independent comparison of NoSQL databases: Cassandra, HBase, MongoDB, Riak,” *Network World*, 2012.

[5] P. Cao, B. Gowda, S. Lakshmi, C. Narasimhadevara, P. Nguyen, J. Poelman, "From BigBench to TPCx-BB: Standardization of a big data benchmark," In *Proceedings of Technology Conference on Performance Evaluation and Benchmarking*, pp. 24-44, 2016.

[6] Cassandra, “The Cassandra Query Language (CQL),” Apache Software Foundation, online: http://cassandra.apache.org/doc/4.0/cql/, accessed August 20, 2018.

[7] Cisco ParStream, "Cisco ParStream: Cisco ParStream Manual," Cisco and/or its affiliates, October 4, 2017. https://www.cisco.com/c/dam/en/us/td/docs/cloud-systems-management/cisco-edge-fog-fabric/1_1_0/Cisco-ParStream-MANUAL-5-0-0.pdf.

[8] Cisco Search, "ParStream", Cisco, https://search.cisco.com/search?query=Cisco%20ParStream&locale=enUS&tab=Cisco, accessed 20.08.2018.

[9] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of 1st ACM symposium on Cloud computing*, pp. 143-154, 2010.

[10] Cisco, Cisco ParStream - Retirement Notification, https://www.cisco.com/c/en/us/obsolete/analytics-automation-software/cisco-ParStream.html, accessed August 20, 2018.

[11] Datastax, "Benchmarking top NoSQL databases: A performance comparison for architects and IT managers," White Paper, Datastax Corporation, Feb. 2013, http://files.meetup.com/7441162/WP-Benchmarking-Top-NoSQL-Databases.pdf, accessed June 06, 2018.

[12] DB-Engines, "DB-Engines Ranking of Wide Column Stores", solid IT gmbh, http://db-engines.com/en/ranking/wide+column+store, accessed June 15, 2017.

[13] S. Edlich, "NoSQL-databases," http://nosql-database.org, accessed June. 15, 2018.

[14] K. Grolinger, W. A. Higashino, A. Tiwari, and M.A.M. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing*, vol. 2, no. 22, 2013.

[15] R. Hecht and S. Jablonski, "NoSQL evaluation: A use case oriented survey," in Proceedings of *International Conference on Cloud and Service Computing*, pp. 336-341, 2011.

[16] Hibernate.org, "Chapter 14. HQL: The Hibernate Query Language", Community Documentation, Red Hat, Inc. online: https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html, accessed 20.08.2018.

[17] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham and C. Matser, "Performance evaluation of NoSQL databases: A case study," in *Proceedings* of *1st Workshop on Performance Analysis of Big Data Systems (PABS '15)*, pp. 5-10, 2015.

[18] J. Kuhlenkamp, M. Klems and O. Röss, "Benchmarking scalability and elasticity of distributed database systems," in *Proceedings of the VLDB Endowment*, vol. 7, pp. 1219-1230, 2014.

H. Kyurkchiev and E. Mitreva, "Performance study of SQL and NoSQL solutions for analytical loads," *Advanced Research in Mathematics and Computer Science*, vol. 49, pp. 49-57, 2014.

[19] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," In *proceedings* of *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 15-19, 2013.

[20] D. Nelubin and B. Engber, "Ultra-high performance NoSQL benchmarking: Analyzing durability and performance tradeoffs," Thumbtack Technology, Inc., White Paper, 2013, http://www.odbms.org/wp-content/uploads/2013/11/NoSQLBenchmarking.pdf, accessed June 15, 2018.

[21] J. Oliveira and J. Bernardino, "NewSQL Databases - MemSQL and VoltDB experimental evaluation," In *Proceedings* of *the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pp. 276-281, 2017.

[22] O. K. Win, Y. Papakonstantinou and R. Vernoux, "The SQL++ unifying semi-structured query language and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases," CoRR, abs/1405.3631, 2014.

[23] Z. Parker, S. Poe and S.V. Vrbsky, "Comparing NoSQL Mongodb to an SQL DB," in *Proceedings of the 51st ACM Southeast Conference*, 2013.

[24] I. Spiegler and R. Maayan, "Storage and retrieval considerations of binary data bases," *Inf. Process. and Manag.*, vol. 21, pp. 233-54, 1985.

[25] B. G. Tudorica and C. Bucur, "A comparison between several NoSQL databases with comments and notes," in *Proceedings of Roedunet International Conference*, 2011.

[26] P. Venkatesh and S. Nirmala, "NewSQL - the new way to handle big data," OpenSource, http://www.opensourceforu.com/2012/01/newsql-handle-big-data, accessed June 15, 2018.

[27] Yahoo! Research, "Yahoo! Cloud Serving Benchmark," April 28, 2010, https://research.yahoo.com/news/yahoo-cloud-serving-benchmark?guccounter=1, accessed 20.08.2018.

[28] 3GPP, "3rd Generation Partnership Project," Technical Report, 3GPP Organizational Partners, June 2018. http://www.3gpp.org/DynaReport/25993.htm.

APPENDIX A: CODE FOR GENERATING RAB REPORTS

The function *generateValues* generates a RAB report with 96 columns and stores it in a table. The simulated RAB reports corresponds to real RAB reports structurally and in content.

```
public void generateValues() {

   Random r = new Random();

   for(int i=0; i<ROWS; i++) {

      Object[] valueArray = new Object[COLUMNS];

      valueArray[0] = inserter_id; //UINT8
      valueArray[1]=ReportID[r.nextInt(ReportID.length)]; //UINT16
      valueArray[2] = r.nextInt(700 - 650) + 650; //UINT32
      valueArray[3] = r.nextInt(); //UINT32
      valueArray[5] = r.nextInt(999 - 100) + 100; //UINT16
      valueArray[6] = r.nextInt(999999 - 10000) + 10000; //UINT32
      valueArray[7] = 15; //UINT8
      valueArray[8] = Long.toString(generateRandomLong(15)); //VARSTRING(16) (IMSI)
      valueArray[9] = r.nextInt(31356 - 29784) + 29784; //UINT16
      valueArray[10] = r.nextInt(9999); //UINT16
      valueArray[11] = r.nextInt(49); //UINT32
      valueArray[12] = r.nextInt(255); //UINT32
      valueArray[13] = r.nextInt(91); //UINT32
      valueArray[14] = Long.toString(generateRandomLong(15)); //VARSTRING(16)
      valueArray[15] = Long.toString(generateRandomLong(15));  //VARSTRING(16) (IMEI)
      valueArray[16] = Integer.toString(r.nextInt(255)) + "." +
      Integer.toString(r.nextInt(255)) + "."+
      Integer.toString(r.nextInt(255)) + "." +
      Integer.toString(r.nextInt(255));  //VARSTRING(16) (IP address)
      valueArray[17] = GenerateIPv6();    //VARSTRING(40)
      valueArray[19] = Long.toString(generateRandomLong(15));  //VARSTRING(16)
      valueArray[20] = r.nextInt(4); //UINT8
      valueArray[21] = r.nextInt(14); //UINT16
      valueArray[22] = r.nextInt(32);
      valueArray[23] = r.nextInt(3058); //UINT16
      valueArray[24] = r.nextInt(3058); //UINT16
      valueArray[25] = r.nextInt(4); //UINT8
      valueArray[26] = r.nextInt(14); //UINT16
      valueArray[27] = r.nextInt(32); //UINT8
      valueArray[28] = r.nextInt(3058); //UINT16
      valueArray[29] = r.nextInt(3058);//UINT16
      valueArray[30] = r.nextInt(6); //UINT16
      valueArray[31] = r.nextInt(266 - 1) + 1; //UINT16
      valueArray[32] = r.nextInt(65534); // UINT16
      valueArray[33] = r.nextInt(65534); //UINT16
      valueArray[34] = r.nextInt(9999999); //UINT32
      valueArray[35] = 244; //UINT16
      valueArray[36] = 7; //UINT16
      valueArray[37] = r.nextInt(65534); //UINT16
      valueArray[38] = r.nextInt(65534); //UINT16
      valueArray[39] = r.nextInt(65534); //UINT16
      valueArray[40] = r.nextInt(65534); //UINT16
      valueArray[41] = r.nextInt(9999999); //UINT32
      valueArray[42] = 244; //UINT16
      valueArray[43] = 7; //UINT16
      valueArray[44] = r.nextInt(65534);  //UINT16
      valueArray[45] = r.nextInt(65534); //UINT16
      valueArray[46] = r.nextInt(9999999 - 1000000) + 1000000; //UINT32
      valueArray[47] = r.nextInt();//UINT32
      valueArray[48] = r.nextInt(254); //UINT8
      valueArray[49] = r.nextInt(254); //UINT8
      valueArray[50] = r.nextInt(65534); //UINT16
      valueArray[51] = r.nextInt(65534); //UINT16
      valueArray[52] = r.nextInt(8);  //UINT8
      valueArray[53] = r.nextInt(999 - 100) + 100;  // UINT16
      valueArray[54] = r.nextInt(2049 - 2048) + 2048; //UINT16
      valueArray[55] = r.nextInt(8);  //UINT16
```

```
      valueArray[56] = r.nextInt(8 - 5) + 5;   //UINT8
      valueArray[57] = r.nextInt(14); //UINT16
      valueArray[58] = r.nextInt(32); //UINT8
      valueArray[59] = r.nextInt(3058);  //UINT16
      valueArray[60] = r.nextInt(3058); //UINT16
      valueArray[61] = r.nextInt(8 - 5) + 5; //UINT8
      valueArray[62] = r.nextInt(99 - 10); //UINT16
      valueArray[63] = r.nextInt(32); //UINT8
      valueArray[64] = r.nextInt(3058);  //UINT8
      valueArray[65] = r.nextInt(3058); //UINT16
      valueArray[66] = r.nextInt(266 - 1) + 1;  //UINT16
      valueArray[67] = r.nextInt(65534); //UINT16
      valueArray[68] = r.nextInt(65534); //UINT16
      valueArray[69] = r.nextInt(9999999); //UINT32
      valueArray[70] = 244; //UINT16
      valueArray[71] = 7; //UINT16
      valueArray[72] = r.nextInt(65534); //UINT16
      valueArray[73] = r.nextInt(65534);  //UINT16
      valueArray[74] = r.nextInt(65534); //UINT16
      valueArray[75] = r.nextInt(65534);  //UINT16
      valueArray[76] = r.nextInt(9999999); //UINT32
      valueArray[77] = 244; //UINT16
      valueArray[78] = 7; //UINT16
      valueArray[79] = r.nextInt(65534); //UINT16
      valueArray[80] = r.nextInt(65534); //UINT16
      valueArray[81] = r.nextInt(7 - 1) + 1; //UINT16
      valueArray[82] = r.nextInt(15); //UINT8
      valueArray[83] = r.nextInt(999999999); //UINT32
      valueArray[84] = r.nextInt(999999999); //UINT32
      valueArray[85] = r.nextInt(999999999); //UINT32
      valueArray[86] = r.nextInt(999999999); //UINT16
      valueArray[87] = r.nextInt(999999999); //UINT16
      valueArray[88] = 244; //UINT16
      valueArray[89] = 7; //UINT16
      valueArray[90] = r.nextInt(999 - 100) + 100; //UINT16
      valueArray[91] = r.nextInt(2049 - 2048) + 2048; //UINT16
      valueArray[92] = r.nextInt(); //UINT16
      valueArray[93] = r.nextInt(); //UINT16
      valueArray[94] = r.nextInt(65534); //UINT16
      valueArray[95] = r.nextInt(65534); //UINT16

      rowsToInsert.add(valueArray);
      generationCount++;
    }

    generationReady = true;
  }
```

AUTHOR BIOGRAPHIES

Petri Kotiranta has graduated from Tampere University of Technology in 2013 and University of Tampere in 2015. In his master's thesis he evaluated the suitability of NoSQL and NewSQL databases for telecommunications data. He has two years of work experience from Nokia Networks where he was working with various software development related tasks. He is currently working in Qvantel as a system specialist with Cassandra database related administration tasks. He is also doing NoSQL related research for his PhD.

Marko Junkkari got Master's Degree in Computer Science from the University of Tampere in 1997. He achieved Licentiate Degree in 2002 and PhD in 2007 in computer science. His research activities cover conceptual structures, graph-based methods, development and integration of data models, query languages, and XML information retrieval. He has been a reviewer in several international journals and a member of several program committees. Since 2006, he has had a position as an Assistant Professor of Data Management in the University of Tampere. His teaching activities include database programming, XML, conceptual modeling, and special topics on data management.